BIASED RANDOM-KEY GENETIC PROGRAMMING

J.F. GONÇALVES AND M.G.C. RESENDE

ABSTRACT. This paper introduces Biased Random-Key Genetic Programming, a new metaheuristic for evolving programs. Each solution program is encoded as a vector of random-keys, where a random-key is a real number randomly generated in the continuous interval $[0, 1)$. A decoder maps each vector of random-keys to a solution program and assigns it a measure of quality. A Program-Expression is encoded in the chromosome using a head-tail representation which is later transformed into a syntax-tree using a prefix notation rule. The artificial simulated evolution of the programs is accomplished with a biased random-key genetic algorithm. Examples of the application of this approach to symbolic regression are presented.

1. INTRODUCTION

Genetic Programming (GP) is an evolutionary metaheuristic inspired by biological evolution to find computer programs that perform a pre-defined user computational task. GP has its origins around 1954 with the evolutionary algorithms applied by Nils Aall Barricelli to evolutionary simulations (Barricelli, 1954). One of the earliest practitioners of the GP methodology was Lawrence J. Fogel who, in 1964, applied evolutionary algorithms to the problem of discovering finite-state automata (Fogel, 1964). The first paper on tree-based genetic programming was presented in Cramer (1985) and was later expanded by John R. Koza, who pioneered the application of genetic programming in various complex optimization and search problems (Koza, 1992; 1994; Koza et al., 1999; 2003).

Traditionally, GP has favored the use of programming languages that naturally embody tree structures (Banzhaf et al., 1998) but several new non-tree representations were suggested and successfully implemented, such as linear genetic programming (LGP) (Brameier and Banzhaf, 2007), gene expression programming (GEP) Ferreira (2001), and Parallel Distributed Graphical Programming (PDGP) (Poli, 1997).

Genetic Programming (GP) is still a young field of research. It attracts a growing research community, and there are many avenues of research yet to be explored. In this paper, we introduce Biased Random-Key Genetic Programming (*BRKGP*), a novel metaheuristic for computer program evolution.

Date: 2018. Cite as J.F. Gonçalves and M.G.C. Resende, "Biased random-key genetic programming," in *Handbook of Heuristics*, R. Martí, P.M. Pardalos, and M.G.C. Resende (Eds.), vol. 1, pp. 23–37, Springer, 2018.

Key words and phrases. Evolutionary algorithm, genetic programming, biased random keys, program evolution.

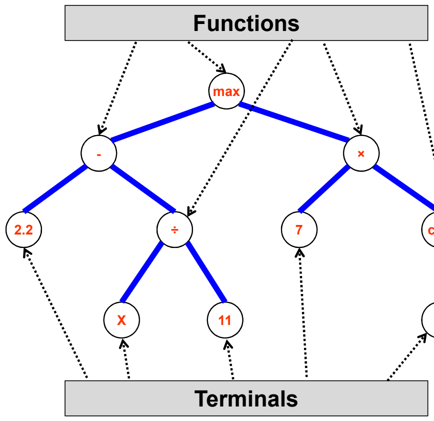

FIGURE 1. Syntax tree of program $max\left(2.2 - \frac{X}{11}, 7 \times cos\,(Y)\right)$.

2. Program Representation

In genetic programming (GP) programs are usually expressed by syntax trees (ST). Figure 1 shows the tree representation of the program

$$max\left(2.2 - \frac{X}{11}, 7 \times cos\,(Y)\right).$$

The variables and constants in the program (X, Y, *2.2*, *11* and *7*) are leaves of the tree. In GP the nodes that correspond to the leaves are called *terminals* and the internal nodes represent the operations ($-$, $\div$, $\times$, cos and max) and are called *functions*. The sets of allowed functions ($\mathcal{F}$) and terminals ($\mathcal{T}$) together form what is called the *primitive set* ($\mathcal{P}$) of a GP system .

In this paper a linear representation of syntax trees is used. This indirect representation, called Program-Expression (PE), encodes a syntax tree as a sequence of elements of the primitive set which will be later translated, according to some predefined rules, into a syntax tree.

The set of rules used to translate a PE into a ST follows a prefix notation convention (*P-rule*). The *P-rule* translates a PE into a ST by reading sequentially, from left to right, each element in the PE and placing it in the bottom-most (first-criteria) and left-most (second-criteria) available node in the partial ST being constructed. Note that if there are no nodes available, in the partially built syntax tree, in which to place an element in the *PE,* then the process stops and the remaining primitive elements in the PE are discarded and are called non-coding elements. Figure 2 demonstrates how the PE

$$\wedge \quad + \quad 1 \quad \div \quad r \quad n \quad n \quad cos \quad Y \quad 17.5$$

can be translated into a ST following the *P-rule*. The first primitive element in the PE is —$\wedge$— and is placed in node 1 (the root of the tree). The second element in the PE is —$+$— and is placed in node 2. The third primitive element in the PE is —1— and is placed in node 4. The fourth primitive element in the PE is —$\div$— and is placed in node 5. This process will be repeated until all elements in the PE are placed or there are no nodes available in the tree in which to place an element.

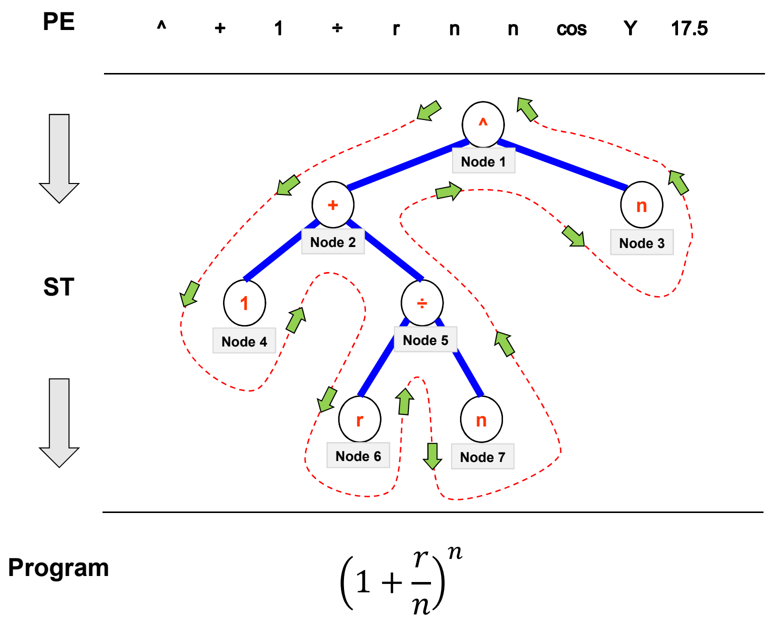

FIGURE 2. Translation of a *PE* into a program using the *P-rule*.

Note that the last three elements of the *PE*, —*cos*—, —*Y*— and —*17.5*—, are not included in the *ST*. The program that corresponds to the final *ST* is

$$\left(1+\frac{r}{n}\right)^n.$$

3. Biased Random-Key Genetic Programming

This section begins with an overview of the proposed Biased Random-Key Genetic Programming *(BRKGP)* methodology. This is followed by a discussion of the biased random-key genetic programming algorithm, including detailed descriptions of the program encoding and decoding and the fitness measure.

3.1. **Overview.**

The *BRKGP* metaheuristic presented in this paper is based on four main components: a biased random-key genetic algorithm (*BRKGA*) (Gonçalves and Resende, 2011), chromosomes, a decoding procedure to translate a chromosome into a syntax tree, and a fitness measure to assess the quality of the resulting programs.

The role of the *BRKGA* is to supply and evolve the chromosomes. A chromosome (genotype) indirectly represents a syntax tree (phenotype) that corresponds to a program. The decoding procedure receives as input a chromosome and in a first phase translates it into a Program-Expression (*PE*). Then, in a second phase the *PE*, obtained in the first phase, is translated into a syntax tree using the *P-rule* described in Section 2.

Figure 3 illustrates the sequence of steps applied by the *BRKGP* to each chromosome.

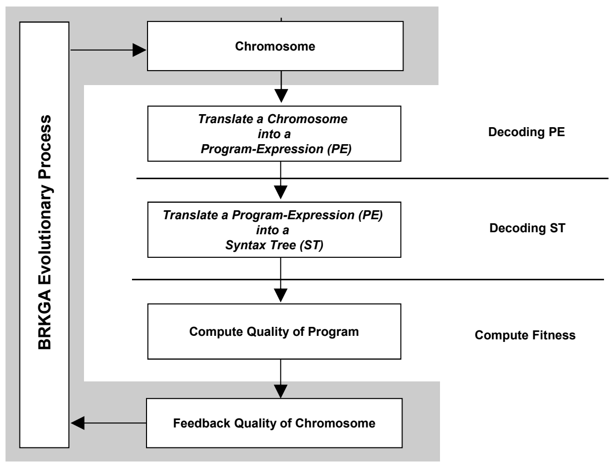
BRKGA Evolutionary Process
Chromosome
Translate a Chromosome into a Program-Expression (PE)
Decoding PE
Translate a Program-Expression (PE) into a Syntax Tree (ST)
Decoding ST
Compute Quality of Program
Compute Fitness
Feedback Quality of Chromosome

FIGURE 3. Architecture of the *BRKGP*.

The remainder of this section describes in detail the biased random-key genetic algorithm, the chromosome structure, the decoding procedure that maps a chromosome into a syntax tree, and the fitness measures.

3.2. **Biased Random-Key Genetic Algorithms.** Genetic algorithms with random keys, or *random-key genetic algorithms* (*RKGA*), were introduced by Bean (1994) for solving sequencing problems. In an *RKGA*, chromosomes are represented as vectors of randomly-generated real numbers in the interval $[0, 1]$. A *decoder* is a deterministic algorithm that takes as input a chromosome and associates it with a solution of the combinatorial optimization problem for which an objective value or *fitness* can be computed.

An *RKGA* evolves a *population* of random-key vectors over a number of *generations* (iterations). The initial population is made up of p vectors each with r random keys. Each component of the solution vector, or random key, is randomly generated, independently, in the real interval $[0, 1]$. After the fitness of each individual is computed by the decoder in generation g, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p - p_e$ *non-elite* individuals. To evolve the population of generation g, a new generation $(g + 1)$ of individuals is produced. All elite individuals of the population of generation g are copied without modification to the population of generation $g + 1$. *RKGAs* implement mutation by introducing *mutants* into the population. A mutant is a vector of random keys generated in the same way that an element of the initial population is generated. Its role is similar to that of mutation in other genetic algorithms (Goldberg, 1989), i.e. to introduce noise into the population and avoid convergence of the entire population to a local optimum. At each generation, a small number p_m of mutants is introduced into the population. With $p_e + p_m$ individuals accounted for in population $g + 1$, $p - p_e - p_m$ additional individuals need to be generated to complete the p individuals that make up population $g + 1$. This is done by producing $p - p_e - p_m$ offspring solutions through the process of *mating* or *crossover*.

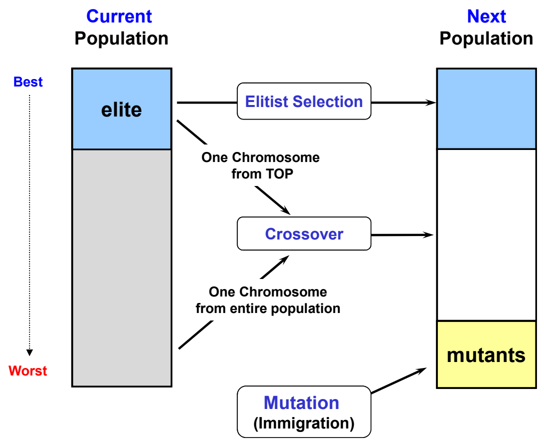

FIGURE 4. Transitional process between consecutive generations.

A *biased random-key genetic algorithm* (Gonçalves and Resende, 2011), or simply *BRKGA*, differs from an *RKGA* in the way parents are selected for mating. While in the *RKGA* of Bean (1994) both parents are selected at random from the entire current population, in a *BRKGA* each element is generated combining a parent selected at random from the elite partition in the current population and one from the rest of the population. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring in the same generation. As in a *RKGA*, *parametrized uniform crossover* (Spears and DeJong, 1991) is used to implement mating in a *BRKGA*. Let ρ_e be the probability that an offspring inherits the vector component of its elite parent. Recall that r denotes the number of components in the solution vector of an individual. For $i = 1, \ldots, r$, the i-th component $c\,[i]$ of the offspring vector c takes on the value of the i-th component $e\,[i]$ of the elite parent e with probability ρ_e and the value of the i-th component $\bar{e}\,[i]$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$.

When the next population is complete, i.e. when it has p individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation. Figure 4 depicts the transitional process between two consecutive generations.

A *BRKGA* searches the solution space of the combinatorial optimization problem indirectly by searching the continuous r-dimensional unit hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

*BRKGA*s have been applied with success to solve many types of combinatorial problems (see, Gonçalves and Almeida (2002), Fontes and Gonçalves (2013), Gonçalves and Resende (2013), Gonçalves and Resende (2015), Gonçalves and Resende (2014), Gonçalves et al. (2014), Gonçalves et al. (2016), and Gonçalves et al. (2015).

A biased random-key genetic algorithm is specified by the parameters, the encoding and decoding of the solutions and by the fitness measure. In the next section the algorithm is specified by first showing how programs are encoded and then decoded and how their fitness evaluation is computed.

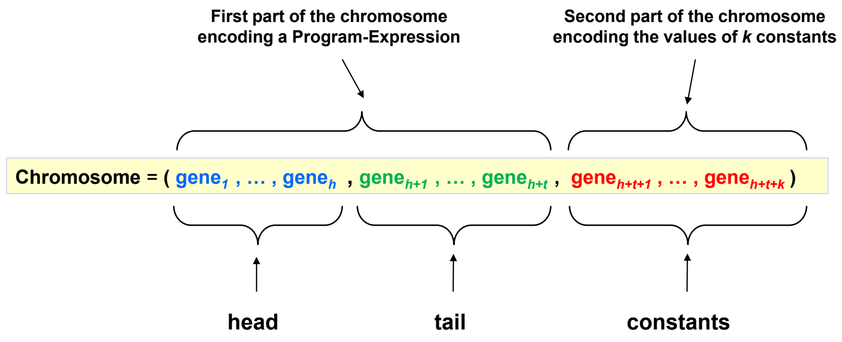
First part of the chromosome encoding a Program-Expression
Second part of the chromosome encoding the values of k constants
Chromosome = (gene1 , … , geneh , geneh+1 , … , geneh+t , geneh+t+1 , … , geneh+t+k)
head
tail
constants

FIGURE 5. Chromosome structure used to encode a *PE* with a vector of $h + t + k$ random keys.

3.3. **Encoding and Chromosome Structure.** A chromosome represents a program and is made of two parts. The first part encodes a *PE* and the second part encodes the values of the constants that can be used in the *PE*. To encode a *PE* a head-tail representation is proposed as in Ferreira (2001; 2006a;b).The head, h, represents the maximum number of internal nodes that can be in the syntax tree (ST) and can contain both functions (elements from set $\mathcal{F}$) and terminals (elements from the set $\mathcal{T}$), whereas the tail, t, represents the number of leaves of the ST and can only contain terminals (elements from the set $\mathcal{T}$). For each problem, the length of the head, h, is chosen, whereas the length of the tail, t, depends on h and on the arity, a, of the function with the most arguments, and is determined by the expression

$$t = h\,(a - 1) + 1. \tag{1}$$

This way of determining the length of the tail guarantees that there will always be enough terminals to create syntactically valid *PE*s (assuming closure amongst the set of primitives).

Assuming that the terminal set includes k constants, then every chromosome supplied by the *BRKGA* encodes a Program-Expression (PE) as a vector of $(h + t + k)$ random keys and Figure 5 depicts the corresponding chromosome structure.

3.4. **Decoding a Chromosome into a *PE*.** The decoding of a chromosome into a Program-Expression (PE) has the following three steps:

- Decoding the head of the *PE*;
- Decoding the tail of the *PE*;
- Decoding the values of the constants.

Let the terminal set $\mathcal{T}$ be divided into the two mutually exclusive sets $\mathcal{V}$ and $\mathcal{K}$ which represent the set of variables and the set of constants, respectively, and let vF, vV, and vK represent vectors containing all the elements in the sets $\mathcal{F}$, $\mathcal{V}$, and $\mathcal{K}$, respectively. Additionally, let nF, nV, and nK represent the number of elements in the vectors vF, vV and vK, respectively.

To better illustrate the various steps in the decoding process, an example based on the information presented in Table 1 is used.

Table 1. Information relative to the example.

$\mathcal{F}=\left\{+,\,-,\,\times,\,\div,\,\surd\right\}$	$\Longrightarrow$	$vF=(+,\,-,\,\times,\,\div,\,\surd)\ nF=5$
$\mathcal{T}=\{X,\,Y\,,Z,\,K_1,\,K_2\}$	$\Longrightarrow$	$vV=\{X,\,Y\,,Z\}\,,\ nV=3\ vK=\{K_1;\,K_2\}\,,\ nK=2$
$h=5,\ a=2$	$\Longrightarrow$	$t=5\times(2-1)+1=6$

```
procedure DecodeHead ($vF$, $vV$, $vK$, $c$, $PE$)
1  for $r = 1, \ldots, h$ do
2      $idx = \lceil c\,[r] \times (nF + nV + nK)\rceil$
       // where $\lceil x\rceil$ is the smallest integer greater or equal to x.
       // ** Decoding $c\,[r] \rightarrow PE\,[r]$ **
3      if $(idx \leq nF)$ then
           // it going to be a function
4          $PE\,[r] = vF[idx]$;
5      else if $(nF < idx \leq nF + nV)$ then
           // it going to be a variable
6          $PE\,[r] = vV[idx - nF]$;
7      else if $(nF + nV < idx \leq nF + nV + nK \leq$then
           // it going to be a constant
8          $PE\,[r] = vK[idx - nF - nV]$;
9      endif
10 end for
end DecodeHead;
```

Figure 6. Pseudo-code for the DecodeHead procedure.

According to Table 1 and the chromosome structure defined in the previous section, the chromosome will have 13 (5+6+2) random-keys. Furthermore, assume that the chromosome being decoded is

$$c=(.25,\ .35,\ .93,\ .75,\ .05,\ .32,\ .67,\ .58,\ .15,\ .26,\ .86,\ .64,\ .43)\,. \tag{2}$$

The decoding of the head of the PE is based on the fact that the head can be made of elements in any of the sets $\mathcal{F}$, $\mathcal{V}$, and $\mathcal{K}$. The mapping of the random-key in position $r = 1, ..., h$ of the chromosome c, $c\,[r]$, into the primitive element in position r of the PE , $PE\,[r]$, is accomplished by the procedure DecodeHead whose pseudo-code is shown in Figure 6.

Table 2 presents the results of the decoding of the head of the PE corresponding to the first five components of the chromosome given in expression (2).

At this point the first five components of the PE will be

$$\times \quad \div \quad K_2 \quad Z \quad + \ .$$

The decoding of the tail of the PE is based on the fact that the tail can only consist of elements in sets $\mathcal{V}$ and $\mathcal{K}$. The mapping of the random-key in position $r = h+1, ..., h+t$ of the chromosome c, $c\,[r]$, into the primitive element in position

TABLE 2. Decoding the Head of the PE.

r	$c[r]$	*idx*	*Head of the PE*
1	0.25	$\lceil 0.25 \times (5+3+2) \rceil = 3$	$vF[3] = \times$
2	0.35	$\lceil 0.35 \times (5+3+2) \rceil = 4$	$vF[5] = \div$
3	0.93	$\lceil 0.93 \times (5+3+2) \rceil = 10$	$vK[10-5-3] = K_2$
4	0.75	$\lceil 0.75 \times (5+3+2) \rceil = 8$	$vV[8-5] = Z$
5	0.05	$\lceil 0.05 \times (5+3+2) \rceil = 1$	$vF[1] = +$

```
procedure DecodeTail ($vV$, $vK$, $c$, $PE$)
1  for $r = h+1, \ldots, h+t$ do
2      $idx = \lceil c[r] \times (nV + nK) \rceil$
       // where $\lceil x \rceil$ is the smallest integer greater or equal to x.
       // ** Decoding $c[r] \rightarrow PE[r]$ **
3      if ($idx \leq nV$) then
           // it going to be a variable
4          $PE[r] = vV[idx]$;
5      else if ($nV < idx \leq nV + nK$) then
           // it going to be a constant
6          $PE[r] = vK[idx - nV]$;
7      endif
8  end for
end DecodeTail;
```

FIGURE 7. Pseudo-code for the DecodeTail procedure.

r of the PE, $PE[r]$, is accomplished by the procedure DecodeTail which has the pseudo-code presented in Figure7.

Table 7 presents the results of the decoding of the head of the PE corresponding to the chromosome given in expression (2).

At this point the components of the PE will be

$$\times \quad \div \quad K_2 \quad Z \quad + \quad Y \quad K_1 \quad Z \quad X \quad Y \quad K_2 \ .$$

This PE is translated with the *P-Rule* into

$$\frac{K_2}{Z} \times (Y + K_1)\,. \tag{3}$$

Note that the last four components in the PE are non-coding elements.

The final step in the decoding process consists in decoding the values of the constants in the terminal set. That is accomplished by using the decoding expression

$$K_i = fk(c[h+t+i], p_1, p_2, ..., p_p) \quad i = 1, ..., nK,$$

TABLE 3. Decoding the tail of the PE.

r	$c[r]$	idx	*Tail of the PE*
6	0.32	$\lceil 0.32 \times (3+2) \rceil = 2$	$vV\,[2] = Y$
7	0.67	$\lceil 0.67 \times (3+2) \rceil = 4$	$vK\,[4-3] = K_1$
8	0.58	$\lceil 0.58 \times (3+2) \rceil = 3$	$vV\,[3] = Z$
9	0.15	$\lceil 0.15 \times (3+2) \rceil = 1$	$vV\,[2] = X$
10	0.26	$\lceil 0.26 \times (3+2) \rceil = 2$	$vV\,[4-3] = Y$
11	0.86	$\lceil 0.86 \times (3+2) \rceil = 5$	$vK\,[5-3] = K_2$

where $fk()$ is a function that accepts as input a random-key, $c\,[r]$, and a set of parameters $p_{1,}\, p_2,\, ..., p_p$, and outputs a real or integer value. The simplest case can be obtained when $fk(c\,[r]) = c\,[r]$, i.e., the value of the constant is equal to the value of the random-key input. Many functions can be used for $fk()$. However, the following functions were used:

(1) **randReal**$(c\,[r]\,,\, l,\, u) = l + c\,[r] \times (\,u - l\,)$ - which generates a real value between l and u;
(2) **randInt**$(c\,[r]\,,\, l,\, u) = \lfloor\, l + c\,[r] \times (u - l\,)\,\rfloor$ - which generates an integer value between the integers l and u.

For the example, assuming $fk(c\,[r]) = c\,[r]$, the values of the constants K_1 and K_2 are, respectively, $c[5+6+1] = 0.64$ and $c[5+6+2] = 0.43$. The final program can be obtained by replacing the constants K_1 and K_2 in expression (3) by 0.64 and 0.43, respectively, i.e.,

$$\frac{0.43}{Z} \times (Y + 0.64)\,. \tag{4}$$

3.5. **Fitness Function.** The objective of the fitness measure is to assign a quality value to each chromosome so that the evolutionary process differentiates the different chromosomes and directs the search to better solutions (chromosomes). Depending on the problem being solved different fitness measures can be used.

Suppose that a symbolic regression problem is to be solved where the best fitting function to a set of points is to be found. In this case the usual measures used in the linear or nonlinear regression can be used, least squares, mean squared error, etc. However, if, for example, a rule to decide when to buy or sell a certain stock in the NYSE is to be discovered, then the quality of rule could be evaluated by using historical data, and the profit obtained with the rule, over a certain number of historical days, could be considered the measure of fitness.

4. EXAMPLES

This section presents two examples in the area of symbolic regression (*SR*) to illustrate the application of *BRKGP*. The Mean Absolute Error (MAE)

TABLE 4. Dataset for *SR*-example 1.

x	y	x	y	x	y
1	10	11	62810	21	806610
2	98	12	88428	22	969958
3	426	13	121186	23	1156946
4	1252	14	162302	24	1369752
5	2930	15	213090	25	1610650
6	5910	16	274960	26	1882010
7	10738	17	349418	27	2186298
8	18056	18	438066	28	2526076
9	28602	19	542602	29	2904002
10	43210	20	664820	30	3322830

TABLE 5. *BRKGP* configuration for the *SR*-example 1.

$\mathcal{F} = \{+, -, \times, \div\}$	$p = 500$
$\mathcal{T} = \{x\}$	$p_e = 100$
$h = 5, 10, 15$	$p_m = 100$
$a = 2$	$\rho_e = 0.85$
$t = h \times (a-1) + 1$	Fitness = Mean Absolute Error (MAE)
	Stopping Criterion = 100 generations

$$MAE = \frac{1}{ND} \sum_{d=1}^{d=ND} |f_d - y_d|$$

is used as fitness measure, where ND is the number of data points, y_d is the value of data point d and f_d is the value obtained by the expression generated by the *BRKGP* for data point d.

4.1. **SR-Example 1.** Table 4 presents thirty data points that are sampled from the function $y = x + 2x^2 + 3x^3 + 4x^4$. *BRKGP* is used to try to discover the polynomial that best fits the data.

The *BRKGP* configuration used for this example is given in Table 5.

Note that, since the best value of h to use is not known, three possibilities for h ($h = 5$, $h = 10$, $h = 15$) are tried. For $h = 5$ and $h = 10$ the *BRKGP* was not able to find the correct polynomial (i.e., $MAE > 0$). However, with $h = 15$ it was able to find the correct polynomial (i.e., $MAE = 0$) after 12 generations. The final

TABLE 6. Dataset for *SR*-example 2.

R	$Area$	R	$Area$
10	314.1592654	21	1385.44236
11	380.1327111	22	1520.530844
12	452.3893421	23	1661.902514
13	530.9291585	24	1809.557368
14	615.7521601	25	1963.495408
15	706.8583471	26	2123.716634
16	804.2477193	27	2290.221044
17	907.9202769	28	2463.00864
18	1017.87602	29	2642.079422
19	20 1256.637061	30	2827.433388

TABLE 7. *BRKGP* configuration for the *SR*-example 2.

$\mathcal{F} = \{+, -, \times, \div\}$	$p = 500$
$\mathcal{T} = \{R, K_1, K_2, K_3\}$	$p_e = 100$
$fk() = \textbf{randInt}(1, 10000)$	p_m= 100
$h = 10$, 15	ρ_e=0.85
$a = 2$	Fitness = Mean Absolute Error (MAE)
$t = h \times (a - 1) + 1$	Stopping Criterion = 100 generations

expression was

$$(((((X + X) \times (X \times X)) + X) \times ((X + X) + (X \div X))) + ((X \times X) \times X))$$

which after being simplified gives the correct polynomial.

4.2. ***SR*-Example 2.** In this example *BRKGP* is used to try to discover the relation between the *Area* of the circle and its radius R that best fits the twenty data points presented in Table 6. The twenty data points were sampled from the function $Area = \pi R^2$.

The *BRKGP* configuration used for this example is given in Table 7. Note that, in this case, three constants K_1, K_2, K_3 are included that will be decoded using $fk() = \textbf{randInt}(1, 10000)$.

For $h = 10$ *BRKGP* found, after 67 generations, the expression

$$((R + ((R \div 689) \div 689)) \times (((9151 \div 4273) \times R) + R))$$

which after being simplified is equivalent to $Area = 3.1415933\, R^2$ and has $MAE =$ 0.00027997.

For $h = 15$ *BRKGP* found, after 82 generations, the expression

$$((R \times (((R + R) + R) \times ((476 + 9787) - (6699 \div 476)))) \div 9787)$$

which after being simplified is equivalent to $Area = 3.1415939\, R^2$ and has $MAE = 0.00051594$.

Note that, in both cases, the value of π was approximated to five decimals.

5. Conclusions

This paper introduced Biased Random-key Genetic Programming, a novel metaheuristic for genetic programming. After introducing how programs are represented using a linear Program-Expression representation, the paper presents the head-tail encoding and explains how the decoding of the chromosome can be accomplished. The paper concludes by illustrating how *BRKGP* can be applied to solve problems using two symbolic regression examples.

Acknowledgement

The first author was supported by project PTDC/EGE-GES/117692/2010 funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology.

References

W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic programming: An Introduction.* Morgan Kaufmann Publishers, San Francisco, CA, 1998.

N.A. Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, 6:45–68, 1954.

J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6:154–160, 1994.

M.F. Brameier and W. Banzhaf. *Linear genetic programming.* Springer Science & Business Media, New York, NY, 2007.

N.L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13:87–129, 2001.

C. Ferreira. Designing neural networks using gene expression programming. In *Applied Soft Computing Technologies: The Challenge of Complexity*, pages 517–535. Springer, 2006a.

C. Ferreira. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence (Studies in Computational Intelligence).* Springer-Verlag New York, Inc., 2006b.

L.J. Fogel. *On the organization of intellect.* PhD thesis, UCLA, 1964.

D.B.M.M. Fontes and J.F. Gonçalves. A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks. *Optimization Letters*, 7:1303–1324, 2013.

D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

J.F. Gonçalves and M.G.C. Resende. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, 21:215–246, 2014.

J.F. Gonçalves and M.G.C. Resende. A biased random-key genetic algorithm for the unequal area facility layout problem. *European J. of Operational Research*, 246:86–107, 2015.

J.F. Gonçalves, J.J.M. Mendes, and M.G.C. Resende. The basic multi-project scheduling problem. In *Handbook on Project Management and Scheduling*, volume 2, pages 667–683. Springer, 2015.

J.F. Gonçalves and J. Almeida. A hybrid genetic algorithm for assembly line balancing. *J. of Heuristics*, 8:629–642, 2002.

J.F. Gonçalves and M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *J. of Heuristics*, 17:487–525, 2011.

J.F. Gonçalves and M.G.C. Resende. A biased random-key genetic algorithm for a 2D and 3D bin packing problem. *International J. of Production Economics*, 145: 500–510, 2013.

J.F. Gonçalves, M.G.C. Resende, and R.F. Toso. An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional*, 34: 143–164, 2014.

J.F. Gonçalves, M.G.C. Resende, and M.D. Costa. A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research*, 23:25–46, 2016.

J.R. Koza. *Genetic programming: On the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

J.R. Koza. *Genetic programming II: Automatic discovery of reusable subprograms.* MIT press, Cambridge, MA, USA, 1994.

J.R. Koza, F.H. Bennett III, D. Andre, and M.A. Keane. *Genetic Programming III:, Darwinian Invention and Problem Solving.* Morgan Kaufmann Publishers, 1999.

J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publisher, 2003.

R. Poli. Evolution of graph-like programs with parallel distributed genetic programming. In *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA)*, pages 346–353, 1997.

W.M. Spears and K.A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

(José F. Gonçalves) INESC TEC and Faculdade de Economia, Universidade do Porto, Porto, Portugal
E-mail address: `jfgoncal@fep.up.pt`

(Mauricio G.C. Resende) Amazon.com and University of Washington, Seattle, WA USA
E-mail address: `mgcr@uw.edu`